D1-82-0219

BOEING SCIENTIFIC RESEARCH LABORATORIES

A Fast Procedure for Generating Normal Random Variables

G. Marsaglia
M. D. MacLaren
T. A. Bray

Mathematics Research

August 1962

D1-82-0219

A FAST PROCEDURE FOR GENERATING NORMAL RANDOM VARIABLES

by

G. Marsaglia, M. D. MacLaren, T. A. Bray

Mathematical Note No. 282

Mathematics Research Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

August 1962

1. Introduction

We will show how to generate normal random variables very rapidly in a computer - for example, at the rate of 10,000 - 15,000 per second in the IBM 7090. The method is suitable for any computer. It has evolved from a number of procedures we have considered in the past, [1] - [3]. The incorporation of successive improvements has led to a procedure which 1) is fairly easy to program, 2) requires little storage, 300 - 400 constants, 3) is very fast - it takes about as long to generate the normal x as the uniform u from which it comes, and 4) is completely accurate, in the sense that in theory the procedure returns a random variable with exactly the required distribution; in practice the result is an approximation influenced only by the capacity (word length) of the computer.

In short, our method is much faster than any we have heard of, and is completely accurate. We recommend that it be used as the basic normal random variable generator in any computer installation. Comparisons of methods, times, storage requirements, etc., are given in Section 4.

2. The procedure for decimal computers

We will go into detail for the decimal case. The method for the binary case is similar and the necessary modifications will be given

in Section 3. Suppose we have a procedure for generating uniform [0,1] random variables u with, say, 8 decimal places. Then probably the fastest way to generate a variable x with absolutely continuous distribution F is to assign a value of x for each of the 10^8 possible u's according to the relation $x = F^{-1}(u)$. The time for generating x would then be the time to generate u, look up the value stored in the location associated with that u, and bring it to the required place. Unfortunately, this fastest procedure would require 10^8 storage locations. Our procedure approaches this ultimate in speed in the following way: 96 - 97% of the time, we use the first few digits of u to locate one of only a few hundred stored values, then we add the last digits of u to that stored value. The other 3 - 4% of the time, we do what is necessary to make the resulting mixture come out right. The average time for the entire procedure still remains close to the ultimate which can be attained by using 10^8 storage locations. The compact storage procedure which enables us to conserve storage space is described in [4].

Now for details of the method. Assume we have a procedure for producing independent uniform [0,1] random variables

$$u, u_1, u_2, u_3, \ldots.$$

The problem is to generate a normal random variable X in terms of

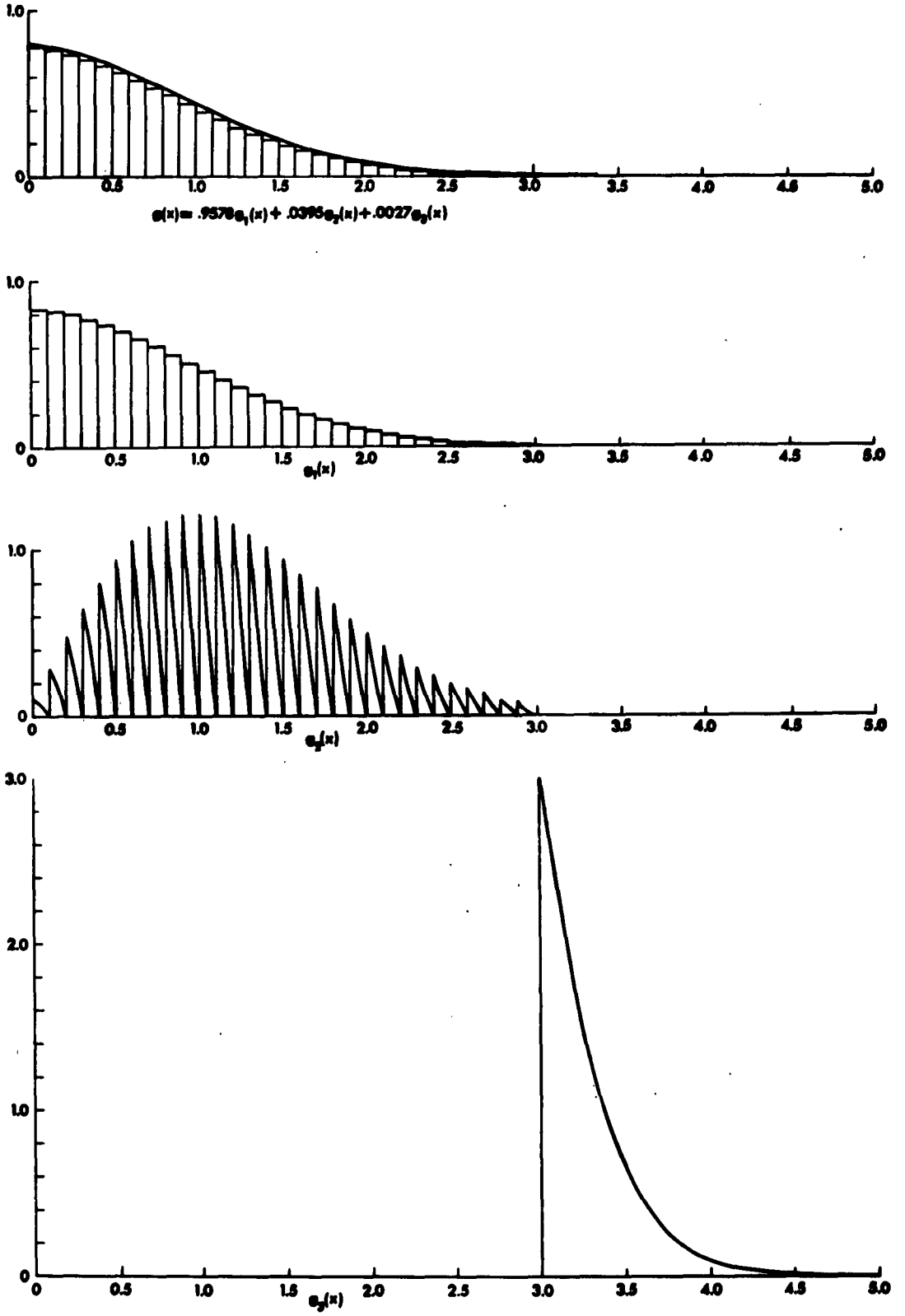

1.0
0
0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
e(x)= .9578e1(x)+ .0395e2(x)+.0027e3(x)
1.0
0
0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
e1(x)
1.0
0
0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
e2(x)
3.0
2.0
1.0
0
0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
e3(x)

Figure 1.

the u's. Let g be the absolute normal density:

$$(1) \qquad g(x) = \frac{2e^{-\frac{1}{2}x^2}}{\sqrt{2\pi}} \qquad x \geq 0$$

The procedures below provide a random variable with density g; a random $\pm$ must be attached at some convenient point in the program, or else, by doubling the storage requirements, the procedure may be modified to handle the standard normal density on $-\infty < x < \infty$.

We write g as a mixture of three densities, as pictured in Figure 1:

$$(2) \qquad g(x) = .9578g_1(x) + .0395g_2(x) + .0027g_3(x).$$

The probabilities displayed in (2) have been rounded to 4 places; they are actually used to 10 places in the program. With probability .9578... we generate a random variable with density g_1, with probability .0395... we generate a random variable with density g_2, and with probability .0027... we generate a random variable with density g_3. The time for g_1 is very short, for g_2 short, and for g_3, long.

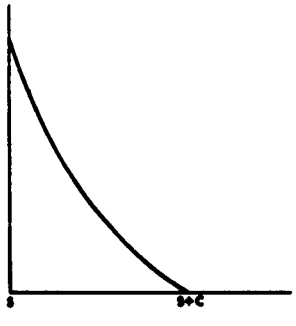
We may produce a variable with density g_1 in the form z + .1u, where z is discrete and u uniform [0,1]. The compact storage

method described in [4] provides z quickly, with modest storage requirement. The altitudes of the rectangles in g_1 are truncated to three decimal places for use in that method, the remaining portion is lumped with the toothlike region above the rectangle and dealt with directly on those infrequent occasions when it is required.

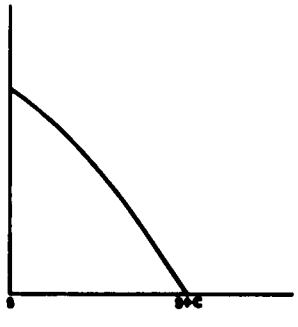
A variable with density g_2, required about 4 percent of the time, is produced as follows: One of the "teeth" from g_2 is selected with appropriate probability, then a random variable with the nearly linear density given by that tooth is generated. The procedure, described in [3], is summarized as follows:

To generate a random variable X with a nearly linear density function $g(x)$, $s \leq x \leq s + c$

such as this

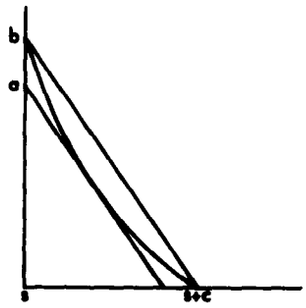


or this

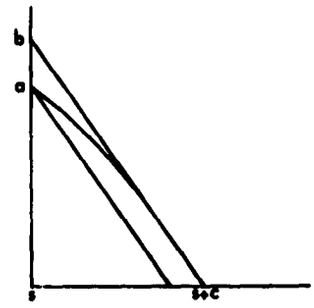


enclose $g(x)$ within two parallel lines,

like this



or this.



Then:

1. Choose independent uniform [0,1] random variables u and v.
2. If $\max(u,v) \leq a/b$ put $X = s + c \min(u,v)$.
3. If not, test: $b|u - v| \leq g[s + c \min(u,v)]$? if yes, put $X = s + c \min(u,v)$; if no, go to step 1 and try again.

In our application of this method, we have $c = .1$ and the test in step 3 may be put in the form

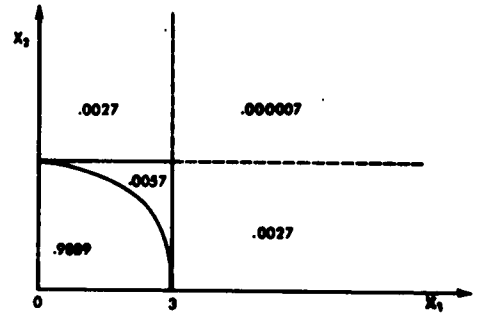
$k|u - v| < e^w - 1$, where $w = -\frac{1}{2}\{[s + .1 \min(u,v)]^2 - [s + .1]^2\}$

and $k = .1(s + .1)$ for $s = 0,.1,.2,\ldots,.9$ and

$k = e^{.1s+.005} - 1$ for $s = 1.0,1.1,1.2,\ldots,2.9$.

The ratios a/b are stored in memory locations 230-259. Most of them are close to 1, so that step 2 of the nearly linear technique will provide X most of the time; only occasionally will an exponential subroutine be required.

Density g_3 comes from the tail of g. It will provide our random variable about 1 time in 400. We then need an absolute normal variable, $|x|$, conditioned by $|x| > 3$. We get it by choosing one of a pair of absolute normal variables (x_1,x_2) lying in the first quadrant but outside the square:



We generate (x_1,x_2) by choosing a normal point outside the quarter circle, rejecting it if it lies inside the square. Thus, if u_1,u_2 are independent, uniform [0,1], conditioned by $u_1^2 + u_2^2 \leq 1$, we put

$$x_1 = u_1\left[\frac{9+2w}{u_1^2+u_2^2}\right]^{\frac{1}{2}}$$

$$x_2 = u_2\left[\frac{9+2w}{u_1^2+u_2^2}\right]^{\frac{1}{2}}$$

where w has the exponential distribution. We may provide w in the form $-\ln(u_1^2+u_2^2)$, since $u_1^2+u_2^2$ is uniform [0,1] and is independent of $\frac{u_1}{u_2}$ (see [5]). Then if (x_1,x_2) lies outside the square, we take whichever of the coordinates that is ≥ 3. If both are ≥ 3, we might possibly store one for future use, but the relative frequency for both ≥ 3, about 1 in 800, is not enough to justify asking for a possible stored value. The measures of the regions are given in the figure, so that $54/(57+54)$, or about 49% of the time, a pair (x_1,x_2) lying outside the quarter-circle will lie outside the square.

Outline of the program: The following outline describes the procedure for a decimal machine, and the flow chart on the opposite page describes the same procedure with a little more detail. Assume that there is a subroutine for producing independent uniform [0,1] random variables $u,u_1,u_2,\ldots$ on demand. Let the constants $C(n)$ be stored in memory locations $n = 0,1,2,\ldots,289$. Let $\delta = 10^{-1}$. Let $u = .d_1d_2d_3d_4d_5\ldots$ be the first u, the d's its decimal digits. Then:

1. If $00 \le d_1d_2 < 79$, put $x = C(d_1d_2) + .0d_4d_5d_6\ldots$
2. If $790 \le d_1d_2d_3 < 940$, put $x = C(d_1d_2d_3 - 771) + .0d_4d_5d_6\ldots$
3. If $.94 \le u < .9973002039$, put $J = 170$ and go to step 5.
4. If $.9973002039 \le u < 1$, form pairs x_1,x_2:

$$x_1 = u_1\left[\frac{3^2 - 2\ln(u_1^2 + u_2^2)}{u_1^2 + u_2^2}\right]^{\frac{1}{2}}$$

$$x_2 = u_2\left[\frac{3^2 - 2\ln(u_1^2 + u_2^2)}{u_1^2 + u_2^2}\right]^{\frac{1}{2}}$$

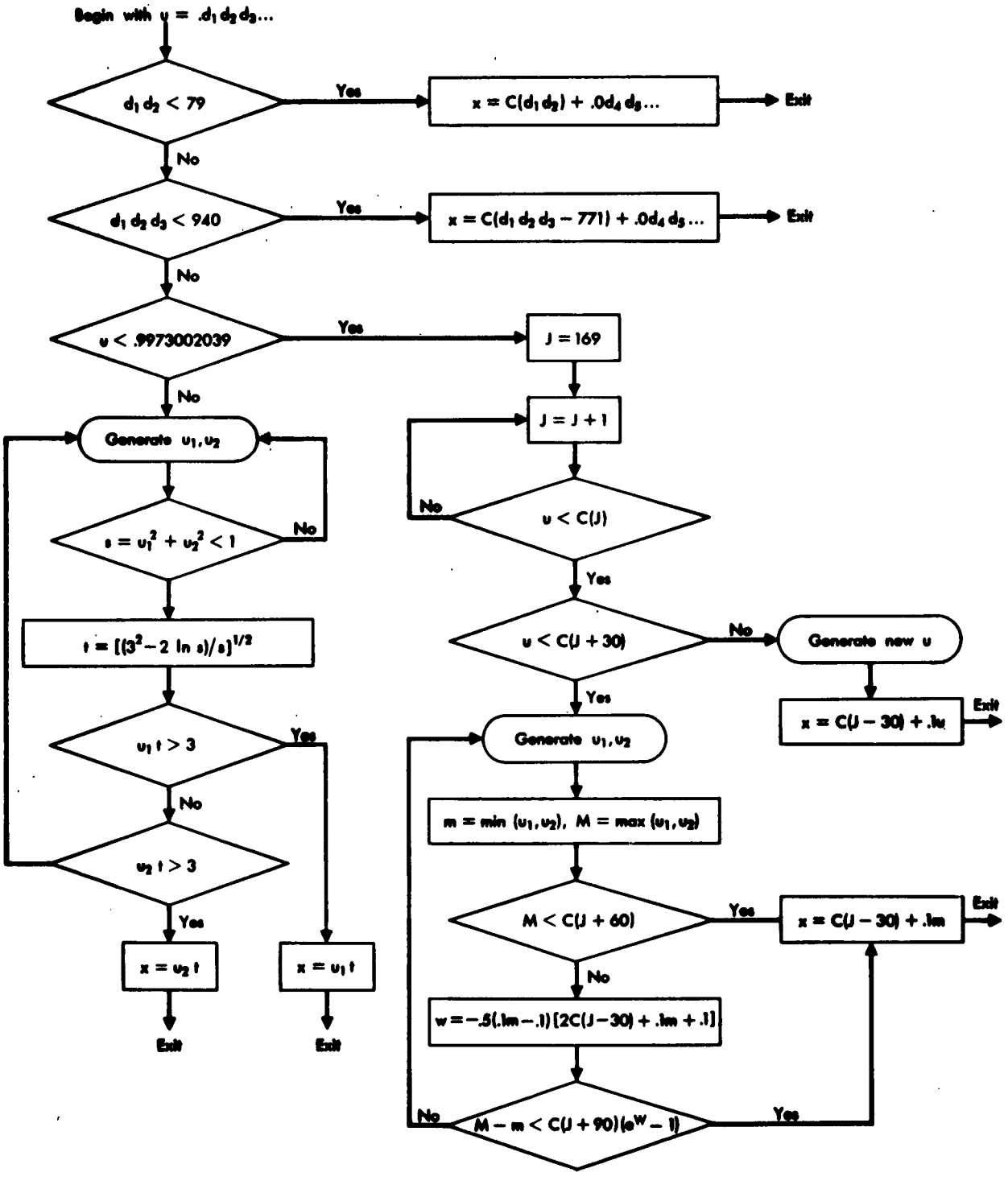
until one of the pair x_1,x_2 is ≥ 3, and let that be x; u_1 and u_2 are conditioned by $u_1^2 + u_2^2 \le 1$.

5. Test: Is $u < C(J)$? If yes, go to step 6. If no, put $J = J + 1$ and repeat step 5.
6. Test: Is $u < C(J + 30)$? If yes, go to step 8. If no, go to step 7.
7. Generate new u and put $x = C(J - 30) + \delta u$.
8. Generate new u_1,u_2. Test: Is $\max(u_1,u_2) < C(J + 60)$? If yes, put $x = C(J - 30) + \delta \min(u_1,u_2)$. If no, let

$$w = -.5[C(J - 30) + \delta \min(u_1,u_2)]^2 - [C(J - 30) + \delta]^2$$

and test: Is $|u_1 - u_2| < C(J + 90)(e^w - 1)$? If yes, put $x = C(J - 30) + \delta \min(u_1,u_2)$. If no, repeat step 8.

To generate a normal variable X in a decimal computer:



Constants for Loading in Memory of a Decimal Machine

Location	Contents										
0	0.2	50	0.5	100	1.6	150	0.5	200	.9422781966	250	.973
1	0.2	51	0.6	101	1.6	151	1.2	201	.9455720777	251	.975
2	0.3	52	0.7	102	1.6	152	1.6	202	.9485514463	252	.974
3	0.3	53	0.7	103	1.6	153	1.7	203	.9511653133	253	.978
4	0.3	54	0.7	104	1.7	154	0.3	204	.9549863293	254	.755
5	0.3	55	0.7	105	1.7	155	1.5	205	.9566914271	255	.970
6	0.3	56	0.7	106	1.7	156	2.0	206	.9604850173	256	.501
7	0.5	57	0.8	107	1.8	157	1.8	207	.9638041343	257	.971
8	0.6	58	0.8	108	1.9	158	2.2	208	.9665717757	258	.968
9	0.6	59	0.9	109	1.9	159	0.2	209	.9689169701	259	.967
10	0.6	60	0.9	110	1.9	160	2.5	210	.9712916782	260	12.5
11	0.6	61	0.9	111	1.9	161	2.3	211	.9742012516	261	8.20523339
12	0.6	62	0.9	112	1.9	162	2.4	212	.9761328124	262	6.91865398
13	0.8	63	1.0	113	1.9	163	2.1	213	.9784228835	263	20.0
14	0.8	64	1.0	114	1.9	164	0.1	214	.9805795259	264	9.03255791
15	0.8	65	1.1	115	1.9	165	2.7	215	.9830652062	265	4.64444483
16	1.0	66	1.1	116	2.0	166	0.0	216	.9842240767	266	6.40863082
17	1.0	67	1.1	117	2.0	167	2.6	217	.9863251515	267	10.0
18	1.5	68	1.2	118	2.0	168	2.8	218	.9871415820	268	11.11111111
19	0.0	69	1.2	119	2.0	169	2.9	219	.9888328519	269	14.2857142
20	0.0	70	1.2	120	2.0	170	.9432165072	220	.9894907759	270	16.66666666
21	0.0	71	1.3	121	2.0	171	.9464092887	221	.9907816118	271	7.51041395
22	0.0	72	1.3	122	2.0	172	.9494969394	222	.9917305989	272	5.57434982
23	0.0	73	1.4	123	2.1	173	.9525783787	223	.9930632865	273	5.22886160
24	0.0	74	1.4	124	2.1	174	.9555567647	224	.9938134106	274	25.0
25	0.0	75	1.5	125	2.1	175	.9584896204	225	.9942625460	275	5.96452440
26	0.1	76	1.6	126	2.1	176	.9613885364	226	.9951108014	276	4.39512015
27	0.1	77	1.7	127	2.1	177	.9641982792	227	.9958055523	277	4.92081328
28	0.1	78	1.8	128	2.1	178	.9667888257	228	.9960778669	278	3.96317864
29	0.1	79	0.0	129	2.2	179	.9693677568	229	.9964138342	279	33.33333333
30	0.1	80	0.4	130	2.2	180	.9719365988	230	.973	280	3.44279563
31	0.1	81	0.4	131	2.2	181	.9744749700	231	.996	281	3.77488448
32	0.1	82	0.7	132	2.2	182	.9769426279	232	.992	282	3.60202892
33	0.2	83	0.9	133	2.3	183	.9792129152	233	.920	283	4.16906566
34	0.2	84	0.9	134	2.3	184	.9812335540	234	.998	284	50.0
35	0.2	85	0.9	135	2.3	185	.9832493731	235	.982	285	3.15925147
36	0.2	86	1.1	136	2.4	186	.9850207959	236	.990	286	100.0
37	0.2	87	1.1	137	2.4	187	.9864483145	237	.996	287	3.29564243
38	0.3	88	1.1	138	2.5	188	.9878069895	238	.985	288	3.03248984
39	0.3	89	1.1	139	2.6	189	.9891104150	239	.959	289	2.91437825
40	0.4	90	1.3	140	0.7	190	.9902073697	240	.942		
41	0.4	91	1.3	141	1.1	191	.9912605179	241	.994		
42	0.4	92	1.3	142	1.3	192	.9922362590	242	.986		
43	0.4	93	1.3	143	0.4	193	.9931582051	243	.985		
44	0.4	94	1.3	144	1.0	194	.9940219494	244	.890		
45	0.4	95	1.3	145	1.9	195	.9948456363	245	.988		
46	0.5	96	1.4	146	1.4	196	.9955013109	246	.980		
47	0.5	97	1.4	147	0.9	197	.9958897393	247	.983		
48	0.5	98	1.6	148	0.8	198	.9962683734	248	.977		
49	0.5	99	1.6	149	0.6	199	.9973002039	249	.843		

3. The procedure for binary computers

The method for binary machines is much the same as for the decimal case; the constants change, of course, and the first four octal digits of u are used to locate a discrete variable, rather than the first three in the decimal case. The binary program requires 456 storage locations (456 base 10 = 707 base 8). Here is the outline for a binary machine; all numbers in the outline are octal except the number of the step:

Let $u = .b_1b_2b_3b_4b_5\ldots$ be a uniform [0,1] random variable, the b's its octal digits. Let the constants $C(n)$ be stored in memory locations $n = 0,1,2,\ldots,707$. Let $\delta = 2^{-4}$.

1. If $00 \leq b_1b_2 < 54$, put $x = C(b_1b_2) + (.b_5b_6\ldots)\delta$
2. If $540 \leq b_1b_2b_3 < 733$, put $x = C(b_1b_2b_3 - 506) + (.b_5b_6\ldots)\delta$
3. If $7330 \leq b_1b_2b_3b_4 < 7571$, put $x = C(b_1b_2b_3b_4 - 7161) + (.b_5b_6\ldots)\delta$
4. If $.7571 \leq u < .776474207403$, put $J = 330$ and go to step 6.
5. If $.776474207403 \leq u < 1$, form pairs x_1,x_2:

$$x_1 = u_1\left[\frac{3^2 - 2\ln u_3}{u_1^2 + u_2^2}\right]^{\frac{1}{2}} \qquad x_2 = u_2\left[\frac{3^2 - 2\ln u_3}{u_1^2 + u_2^2}\right]^{\frac{1}{2}}$$

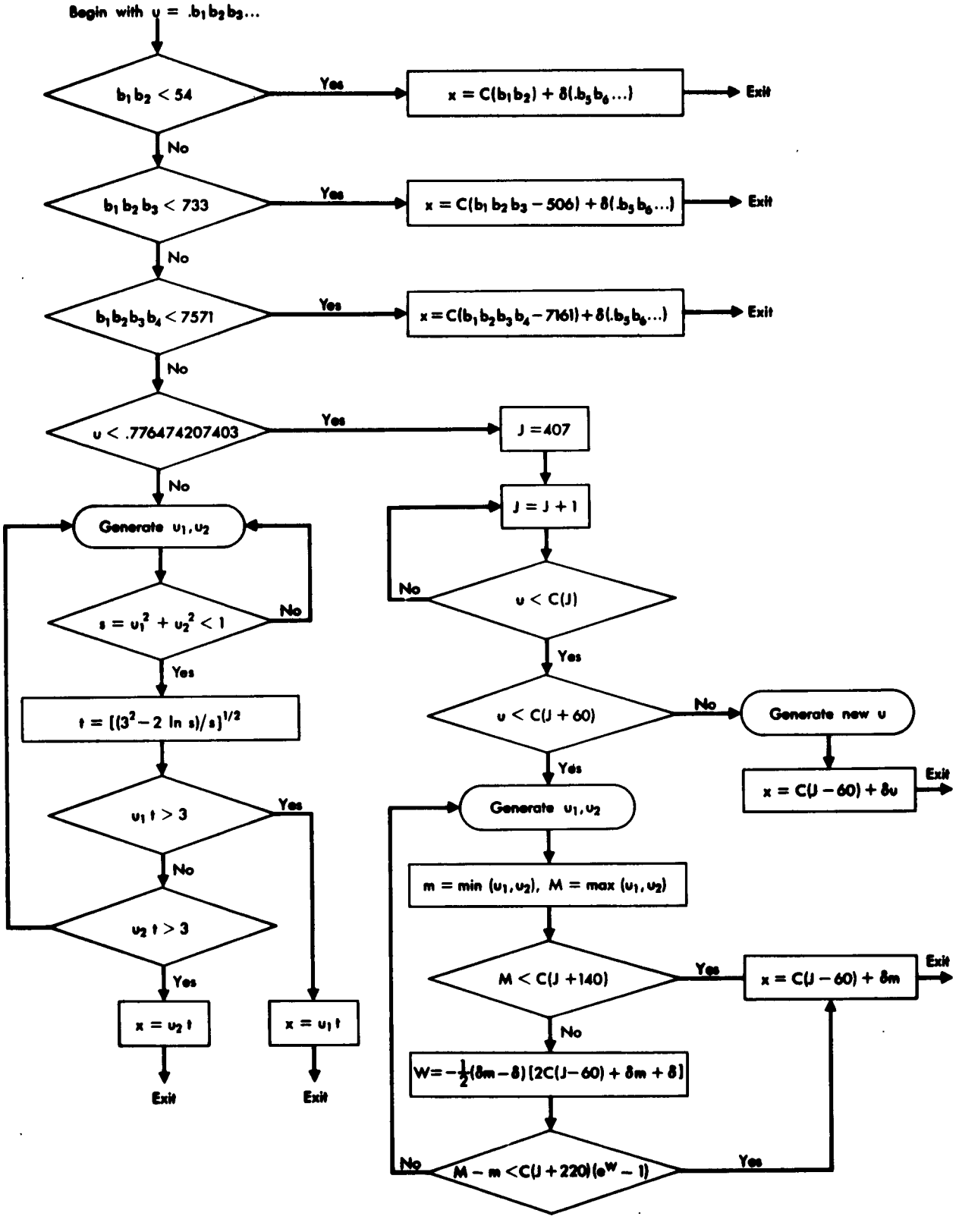
until one of the pair x_1,x_2 is ≥ 3, and let that be x; u_1 and u_2 are conditioned by $u_1^2 + u_2^2 \leq 1$.

6. Test: Is $u < C(J)$? If yes, go to step 7. If no, put $J = J + 1$ and repeat step 6.
7. Test: Is $u < C(J + 60)$? If yes, go to step 9. If no, go to step 8.
8. Generate new u and put $x = C(J - 60) + \delta u$.
9. Generate new u_1,u_2. Test: Is $\max(u_1,u_2) < C(J + 140)$? If yes, put $x = C(J - 60) + \delta \min(u_1,u_2)$. If no, let

$$w = -.5[C(J - 60) + \delta \min(u_1,u_2)]^2 - [C(J - 60) + \delta]^2$$

and test: Is $|u_1 - u_2| < C(J + 220)(e^w - 1)$? If yes, put $x = C(J - 60) + \delta \min(u_1,u_2)$. If no, repeat step 9.

To generate a normal variable X in a binary computer:



Constants for Loading in Memory of a Binary Machine

Location Contents

00	0.00	40	0.34	100	1.00	140	2.34	200	1.44	240	0.74	300	2.40	340	1.34	400	2.64
01	0.00	41	0.44	101	1.00	141	0.00	201	1.44	241	1.00	301	2.40	341	1.24	401	2.74
02	0.00	42	0.44	102	1.04	142	0.04	202	1.44	242	1.00	302	2.40	342	1.30	402	0.50
03	0.04	43	0.50	103	1.04	143	0.24	203	1.54	243	1.04	303	2.40	343	0.54	403	1.34
04	0.04	44	0.50	104	1.04	144	0.24	204	1.60	244	1.04	304	2.40	344	1.44	404	0.14
05	0.04	45	0.54	105	1.30	145	0.24	205	1.60	245	1.14	305	2.40	345	0.40	405	2.04
06	0.10	46	0.54	106	1.40	146	0.24	206	1.60	246	1.20	306	2.44	346	1.40	406	2.40
07	0.10	47	0.60	107	1.40	147	0.24	207	1.60	247	1.20	307	2.44	347	0.44	407	2.20
10	0.14	50	0.74	110	1.40	150	0.34	210	1.64	250	1.20	310	2.44	350	0.30	410	.757560405673
11	0.14	51	1.00	111	1.40	151	0.40	211	1.64	251	1.24	311	2.44	351	0.34	411	.760234424060
12	0.14	52	1.04	112	1.40	152	0.40	212	1.70	252	1.24	312	2.44	352	1.50	412	.760676236473
13	0.20	53	1.10	113	1.40	153	0.40	213	1.70	253	1.24	313	2.50	353	1.60	413	.761334040001
14	0.20	54	0.30	114	1.44	154	0.40	214	1.70	254	1.24	314	2.50	354	1.54	414	.761757765014
15	0.20	55	0.30	115	1.50	155	0.40	215	1.74	255	1.24	315	2.50	355	0.24	415	.762374711576
16	0.40	56	0.30	116	1.50	156	0.54	216	1.74	256	1.34	316	2.50	356	1.74	416	.763001641156
17	0.40	57	0.30	117	1.50	157	0.60	217	2.04	257	1.40	317	2.54	357	1.64	417	.763403057403
20	0.60	60	0.30	120	1.50	160	0.64	220	2.04	260	1.40	320	2.54	360	0.20	420	.764003230622
21	0.64	61	0.34	121	1.50	161	0.74	221	2.10	261	1.54	321	2.54	361	1.70	421	.764374663551
22	0.64	62	0.34	122	1.50	162	1.00	222	2.20	262	1.54	322	2.60	362	2.10	422	.764761323555
23	0.70	63	0.34	123	1.54	163	1.04	223	2.24	263	1.60	323	2.60	363	2.00	423	.765341272422
24	0.70	64	0.44	124	1.54	164	1.10	224	2.30	264	1.60	324	2.64	364	0.10	424	.765671312356
25	1.14	65	0.44	125	1.54	165	1.10	225	0.00	265	1.70	325	2.64	365	2.34	425	.766220112235
26	1.20	66	0.44	126	1.54	166	1.10	226	0.14	266	1.70	326	2.70	366	2.14	426	.766540571732
27	1.24	67	0.50	127	1.64	167	1.14	227	0.14	267	1.70	327	2.74	367	0.04	427	.767057376151
30	1.30	70	0.50	130	1.64	170	1.14	230	0.14	270	2.04	330	0.74	370	2.30	430	.767362761001
31	1.34	71	0.74	131	1.74	171	1.14	231	0.14	271	2.04	331	1.10	371	2.60	431	.767661723253
32	0.10	72	0.74	132	2.00	172	1.20	232	0.20	272	2.20	332	0.60	372	0.00	432	.770147332111
33	0.24	73	0.74	133	2.00	173	1.20	233	0.34	273	2.20	333	0.70	373	2.24	433	.770433574421
34	0.24	74	0.74	134	2.00	174	1.24	234	0.34	274	2.20	334	1.04	374	2.54	434	.770705221531
35	0.30	75	0.74	135	2.10	175	1.40	235	0.64	275	2.20	335	1.14	375	2.44	435	.771146047651
36	0.30	76	1.00	136	2.14	176	1.44	236	0.70	276	2.24	336	0.64	376	2.50	436	.771401242447
37	0.34	77	1.00	137	2.14	177	1.44	237	0.70	277	2.24	337	1.00	377	2.70	437	.771627054774

Constants for Loading in Memory of a Binary Machine (Continued)

Location	Contents									
440	.772052635207	500	.763727170426	540	.774625742514	600	.720305	640	12.70512773444	700	5.11022055440
441	.772270311551	501	.764350307335	541	.774665775115	601	.772274	641	13.7751302113	701	4.71724514567
442	.772467634533	502	.764731761350	542	.775241167360	602	.770652	642	13.323750754 7	702	27.2135056427
443	.772665647261	503	.765325542625	543	.775574501203	603	.771260	643	25.2525252525	703	12.3153453106
444	.773045535544	504	.765626262733	544	.775753645433	604	.654426	644	11.4302476301	704	100.000000000
445	.773222754052	505	.766170332156	545	.776210076455	605	.767227	645	34.3434343435	705	7.11613341352
446	.773356736057	506	.766520025576	546	.776310245131	606	.770652	646	11.7517535166	706	5.6530574 3000
447	.773504033235	507	.767056242601	547	.776466327471	607	.601014	647	31.4631463146	707	6.41506050024
450	.773627400631	510	.767313406006	550	.776764	610	.767636	650	44.4444444445		
451	.773751067050	511	.767641337232	551	.776355	611	.766213	651	40.0000000000		
452	.774061110732	512	.770133535457	552	.770244	612	.400000	652	11.1264167105		
453	.774170242040	513	.770372353604	553	.774733	613	.770244	653	10.3736103657		
454	.774275155254	514	.770672114354	554	.776764	614	.766213	654	10.6426544257		
455	.774402002413	515	.771115102463	555	.775747	615	.766621	655	52.5252525253		
456	.774502233714	516	.771330110360	556	.772702	616	.766621	656	7.50620734567		
457	.774602052000	517	.771611062622	557	.777372	617	.765177	657	10.1400214133		
460	.774647461226	520	.772011173326	560	.774733	620	.765605	660	63.1463146315		
461	.774707157165	521	.772247442063	561	.775747	621	.765177	661	7.71641453111		
462	.775262222404	522	.772415035317	562	.775341	622	.762540	662	6.7347374 4531		
463	.775620521736	523	.772637451342	563	.765605	623	.774325	663	7.30631455432		
464	.776052124043	524	.772770747563	564	.773717	624	.703045	664	125.252525253		
465	.776237170444	525	.773124460000	565	.753412	625	.771260	665	5.77466772305		
466	.776362512763	526	.773336707062	566	.774325	626	.767227	666	6.56205271365		
467	.776474207403	527	.773431174735	567	.757065	627	.770244	667	200.000000000		
470	.757467472603	530	.773571235677	570	.741217	630	20.0000000000	670	6.12270336161		
471	.760141624142	531	.773656531334	571	.746723	631	15.2600345420	671	5.21066362026		
472	.760610704577	532	.773772067714	572	.773311	632	23.5423542354	672	400.000000000		
473	.761264207417	533	.774155267577	573	.772702	633	21.0421042104	673	6.25540044013		
474	.761721150613	534	.774223170530	574	.773311	634	16.1045753441	674	5.31431256077		
475	.762333662456	535	.774341036133	575	.732071	635	14.5047100265	675	5.53521465767		
476	.762757073574	536	.774441106310	576	.771666	636	22.2222222222	676	5.42310106110		
477	.763371074044	537	.774523200672	577	.772274	637	17.0124036200	677	5.01237714432		

4. Remarks

The fast parts of the program, steps 1 - 2 for the decimal and 1 - 3 for the binary, are used most of the time. They should be written in machine language, with care. A procedure for assigning a random $\pm$ must be incorporated, too. The remaining steps can be written in FORTRAN or some such language with little effect on the average running time.

We have written programs based on this procedure for the IBM 1620 (decimal), and the IBM 7090 (binary) machines. The programs are written as standard FORTRAN function subprograms with the necessary linkages, setting index registers, returning X in normalized floating form, etc. In the IBM 7090, each call to the subroutine for a normal variable requires about 50 cycles, which gives a rate of about 10,000 per second. If the procedure is incorporated in a larger program and not written as a standard subroutine, then rates of about 15,000 per second are possible. The standard subroutine for the IBM 7090 requires about 1,300 storage locations, including the space for the constants and for the instructions.

The IBM 1620 decimal machine is quite a bit slower. The above procedure, written as a standard FORTRAN subprogram for that machine, with necessary linkages, returning X in normalized floating point form, etc., takes about 20 milliseconds. The constants and the instructions require about 4,000 core storage positions.

REFERENCES

[1] G. Marsaglia, "Expressing a random variable in terms of uniform random variables", Annals of Math. Stat., Vol. 32, No. 3, September 1961, p. 894.

[2] G. Marsaglia, "Procedures for generating normal random variables, II", Boeing Scientific Research Laboratories Document D1-82-0141, November 1961.

[3] G. Marsaglia, "Random variables and computers", Transactions of the Third Prague Conference, 1962 (To appear).

[4] G. Marsaglia, "Generating discrete random variables in a computer", Comm. Assoc. Comp. Mach.,Vol. 6, No. 1, January 1963.

[5] G. Marsaglia, "Improving the polar method for generating a pair of normal random variables", Boeing Scientific Research Laboratories Document D1-82-0203, September 1962.